Regularization and nonlinearities for neural language models: when are they needed?

Marius Pachitariu
Gatsby Computational Neuroscience Unit
University College London, UK
marius@gatsby.ucl.ac.uk

Maneesh Sahani
Gatsby Computational Neuroscience Unit
University College London, UK
maneesh@gatsby.ucl.ac.uk

Abstract

We show that a recently proposed regularization method called random dropouts works well for language models based on neural networks when little training data is available. Random dropout regularization involves adding a certain kind of noise to the likelihood function being optimized and can be interpreted as a variational approximation to a new class of generative models. We also introduce a simple linear NN model where the nonlinearity of the RNN is removed and we restrict the recurrent matrix to be diagonal. The hidden units in this model compute filtered projections of the input, hence the name FPNN. We show that FPNNs are state-of-the-art language models on the Penn Corpus when properly regularized with random dropouts and column normalization. The nonlinear type of RNN we consider uses rectifier units. Despite their highly nonlinear nature, RNNs are not better language models than FPNNs, even on a large dataset where regularization no longer matters. However, when modelling language at the character-level, FPNNs do not work well, while the RNN trained with stochastic gradient descent achieves similar results to the multiplicative RNN trained with Hessian-Free optimization. GPU training time for the SGD trained RNN is however 50 times less than the training time for HF-trained M-RNN.

1 Neural language modelling

The main paradigm shift in language modelling more than 20 years ago moved the field from rule-based systems to statistical, learned models. The most popular such models are based on frequency statistics of short sequences of words, called N-grams. Various smoothing techniques were proposed to solve the fundamental problem of language modelling: most words and combinations of words appear very rarely in language. In fact, from the point of view of n-gram techniques, language modelling is fundamentally a smoothing or in other words a regularization problem. More recently, models based on neural networks (NNLMs) have been shown to result in better representations of language due (perhaps) to their lower dimensional parametrization. Recurrent NN LMs are a subclass of NN LMs that achieve even better performance with a clever parametrization of the predictive likelihood function through a recurrent neural network.

However, to yield top perplexity scores, the neural network language models currently still need to be combined with N-gram based models, caching techniques and averaged over an ensemble of different models as shown in [1]. Furthermore, the RNN based LMs already require very long training times individually and can be slow at run time if the average over an ensemble needs to be computed. Here we show that random dropout based regularization improves the performance of RNN LMs. Furthermore, a simpler model that we propose here, the filtered projections NN (FPNN), achieves equal performance to the nonlinear RNN when both are regularized in this manner. The FPNN is similar to the log-bilinear language model [2] (LBL) and can also be seen as a learned Long Short Term Memory (LSTM) network [3]. To solve the problem of decaying gradients in

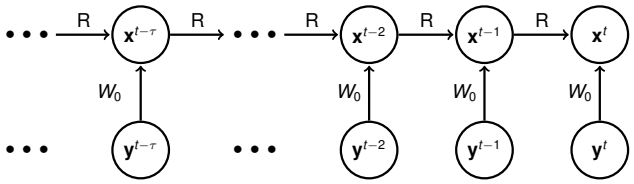

Figure 1: Graphical representation of the function computed by a recurrent neural network.

learning RNNs, special units in the LSTM have a connection strength of 1 to themselves and no connections to the rest of the network. Our FPNN is composed exclusively of these special LSTM units, with the generalization that the self-connection strength can be anything between -1 and 1 and it is learned together with the other parameters of the FPNN. In practice, the FPNN learns to incorporate information from very large contexts of up to 50-100 words, while also capturing local information.

2 Recurrent neural networks and backpropagaton through time

Standard recurrent neural networks are functions of input data. We consider sequential data that comes in the form of discrete tokens, such that the tokens may for example be either characters or words. An RNN function takes the following form, depicted graphically in figure 1:

$$\mathbf{x}_t = f\left(\mathbf{W}\ \ \mathbf{y}_t + \mathbf{R}\ \ \mathbf{x}_{t-1}\right),$$

where $\mathbf{y}_t$ is the data and $\mathbf{x}_t$ is the representation computed by the RNN. In the case of language modelling, $\mathbf{y}_t$ is a one-hot encoding of the token in position t, meaning $\mathbf{y}_t$ is a vector of mostly zeros with just a one in the position of the active token. We call $\mathbf{W}$ the encoding matrix and $\mathbf{R}$ the recurrent or transformation matrix. f is typically taken to be a sigmoidal nonlinearity such as the logistic $\sigma = 1/(1+exp(-x))$. It is generally believed that such strong nonlinearities are necessary to model the apparently complicated dependencies in real-world data. The disadvantages of sigmoidal nonlinearities will become apparent when we consider the optimization problem below. To make a statistical language model out of an RNN, we define a soft-max probability over the tokens in the sequence, such that this probability depends only on the representation $\mathbf{x}_t$ computed by the RNN:

$$\mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{y}_{t-1}, ...\right) \propto \exp(\mathbf{y}_{t+1}^T \mathbf{Z}\mathbf{x}_t), \tag{1}$$

where $\mathbf{Z}$ is a decoding matrix. We propose the terminology of encoding, transformation and decoding matrices for $\mathbf{W}, \mathbf{R}$ and $\mathbf{Z}$ due to the similarity with methods based on autoencoders, RBMs or sparse coding for static data. To obtain the likelihood of a full sequence we multiply together the conditional probabilities defined by equation 1 for every index T from 1 to the length of the sequence.

The likelihood of the RNN LM can be optimized by gradient descent. Notice that in order to learn $\mathbf{W}$ and $\mathbf{R}$ the gradient has to be backpropagated through time (BPTT) over successive activations of $\mathbf{x}_t$. The intermediate gradients at $\mathbf{x}_t$ which we call $\mathbf{D}_t$ can be computed incrementally in the reverse-time direction

$$\mathbf{D}_t = f' \circ \left(\mathbf{Z}^T\ \ \frac{\partial \mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{x}_t\right)}{\partial \mathbf{y}_{t+1}} + \mathbf{R}^T\ \ \mathbf{D}_{t+1}\right), \tag{2}$$

where $\circ$ is elementwise multiplication. This backward pass for BPTT has a similar functional form to the forward pass of equation 1 and the same computational complexity. It has been observed early on in [4] that during training the contribution to the gradient of W and R from future times tends to vanish as it is backpropagated through several time steps. This can be understood by considering the Jacobians $\partial \mathbf{x}_t / \partial \mathbf{x}_{t-\tau}$ of the transformations which the RNN is computing sequentially, and noticing that two main effects alter the size of the gradient/Jacobian. First, the transformation matrix itself generally has eigenvalues less than 1 in absolute value in order to be stable. Consequently, the projection of $\mathbf{x}^T$ on the eigenvectors of $\mathbf{R}$ decays exponentially over time steps, both in the forward direction when computing the likelihood and in the backward direction when computing the gradients, because $\mathbf{R}$ and $\mathbf{R}^T$ have the same eigenvalues. The second effect comes from the elementwise nonlinearity which further multiplies the gradient $\mathbf{D}_t$ elementwise by the gradient of

f. Typically during learning the sigmoid saturates either at 0 or 1 where the derivative f' is 0, which drastically reduces the contribution to the gradient from future time steps. We hypothesized that this second effect has a much greater influence on BPTT than the eigenvalues of the transformation matrix $\mathbf{R}$. In fact, with linear RNNs where f is just the identity, we find that the network easily learns matrices $\mathbf{R}$ with 20% of their eigenvalues larger than 0.9.

2.1 RNN with rectifier units

To model nonlinear and complicated distributions of sequential data a version of RNN in which the nonlinearity f has been replaced with a rectifier function of the form $f(x) = \max(0, x)$. Unlike the sigmoidal nonlinearity, rectifier functions have derivative 1 for positive input which means they fully propagate the gradient in equation 2. This architecture is also currently and independently investigated by [5], though we were not aware of their results when we ran our own experiments. We found that the standard rectifier nonlinearity they use there is relatively unstable during learning. Furthermore, the results reported in [5] for this architecture are well behind the state-of-the-art character level results reported in [6] with the Hessian-Free optimized M-RNN of [7] (see table 2a), and only approximately match a simple unigram model at the word-level. We adopted instead a smoothed version of the rectifier nonlinearity which is differentiable everywhere and still 0 when the input is negative $f(x) = \max(0, x - a \tanh x/a)$ and found that this simple smooth nonlinearity can be used stably with large learning rates. When a is made small this function approaches the standard rectifier nonlinearity. The H-RNN can model highly nonlinear distributions of sequences as shown by its performance on the character-level modelling task in the Results section. We also used the RNN in word-level experiments. There we reverted back to the standard rectifier nonlinearity, as we did not observe the same instabilities.

2.2 Linear RNN

We also considered completely giving up the nonlinearity f and replacing it with the identity. Again this results in complete propagation of the gradient in equation 2, for both positive and negative inputs. The resulting predictive distributions $\mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{y}_t, ...\right)$ fall into the class of generalized linear models (GLM). This is because the FPNN computes a linear function $\mathbf{x}_t$ of the previous observations before passing $\mathbf{y}_{t+1}^T \mathbf{Z}\mathbf{x}_t$ through the softmax nonlinearity. Unlike standard GLM models however, the FPNN is much more compactly parametrized. The log-bilinear LM of [2] has a similar GLM parametrization, however it lacks the long timescales in $\mathbf{x}_t$ which the FPNN can generate. As we will see in the results section, the FPNN is perfectly able to model the complex sequential patterns in language at the word-level. In fact, we found that restricting $\mathbf{R}$ to be diagonal offers similar performance to the FPNN with full $\mathbf{R}$, so for all experiments reported here we used a diagonal $\mathbf{R}$.

The linear contribution to $\mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{y}_t, ...\right)$ before the soft-max nonlinearity can be rewritten as $\mathbf{y}_{t+1}^T \mathbf{Z}\mathbf{R}^\tau \mathbf{W}\mathbf{y}_{t-\tau}$, regardless of whether $\mathbf{R}$ is diagonal or not. The LBL model of [2] uses a different parametrization where $\mathbf{R}^\tau$ is replaced with arbitrary interaction matrices $\mathbf{D}_\tau$. In fact, just like we do here, [2] use diagonal matrices $\mathbf{D}_\tau$. There are several advantages to using $\mathbf{D}_\tau = \mathbf{R}^\tau$. The first and most important is the implicit inductive bias the FPNN has for long timescales: words separated by several other words have an interaction term that only depends weakly on the length of the separation because $\mathbf{R}^\tau$ and $\mathbf{R}^{\tau+1}$ are similar for entries of $\mathbf{R}$ close to 1. In the LBL model the interaction terms are entirely different at different delays τ and $\tau + 1$. Consequently the FPNN can generalize information learned at one delay τ to all other delays. For self-connections in $\mathbf{R}$ close to 1, the hidden units of the FPNN can be thought of as representing the topics of text. In fact, a recent state-of-the-art document topic model uses a similar parametrization of a neural network, there called NADE (neural autoregressor density estimation). NADE is the same model as the FPNN with self-connections of 1 but with an additional nonlinearity in front of $\mathbf{x}_t$ before multiplication with the matrix $\mathbf{Z}$. To model bag-of-words representations of documents, NADE assumes a random ordering of the words in each document.

We also considered a dynamic version of the FPNN, where the parameters of the model are dynamically adapted at test time in the direction of the gradient with respect to the new observations. We found modest but significant improvements, of a similar magnitude as those reported in [1].

3 Regularization with random dropouts and column normalization

We found that on small corpora like Penn, the FPNN was still able to overfit severely despite its relatively low-dimensional parametrization (compared to N-grams). On such problems, we applied the regularization method proposed by [8] and called there random dropout. The idea is to introduce noise into the function computed by the neural network so as to make the network more robust to test examples never seen before. The intuition provided by [8] is that the noise prevents units in the neural network from co-adapting their weights to overfit the training data. To avoid introducing instabilities into the recurrent part of the LM, we added the noise only on the decoding portion of the model. We also provide here a novel variational interpretation of random dropouts which helps us interpret this method differently.

Formally, define η to be a vector of Bernoulli random variables with probabilities 0.5 and length the size of the RNN. Then $\bar{\mathbf{x}}_t = \mathbf{x}_t \circ \eta$, and $\bar{\mathbf{x}}$ replaces $\mathbf{x}$ in equation 1 so that $\mathrm{P}_\eta\left(\mathbf{y}_{t+1}|\eta, \mathbf{y}_t, \mathbf{y}_{t-1}, ...\right) \propto \exp(y_{t+1}^T Z \bar{\mathbf{x}}_t)$. We define

$$\mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{y}_{t-1}, ...\right) = \langle \mathrm{P}_\eta\left(\mathbf{y}_{t+1}|\eta, \mathbf{y}_t, \mathbf{y}_{t-1}, ...\right)\rangle_\eta$$

Because each P_η is a normalized probability, so is the average over all η-s. This is a different generative model than the standard RNN and as we will see it is more well suited to model the types of regularities present in language. Consequently it overfits less to the training data. For training we will use a tractable lower bound on the likelihood of the noisy RNN model. The bound follows from a simple application of Jensen's inequality

$$\begin{aligned}\log \mathrm{P}\left(\mathbf{y}_{t+1}|\mathbf{y}_t, \mathbf{y}_{t-1}, ...\right) &= \log\langle \mathrm{P}_\eta\left(\mathbf{y}_{t+1}|\eta, \mathbf{y}_t, \mathbf{y}_{t-1}, ...\right)\rangle_\eta \\ &\geq \langle \log\langle \mathrm{P}_\eta\left(\mathbf{y}_{t+1}|\eta, \mathbf{y}_t, \mathbf{y}_{t-1}, ...\right)\rangle_\eta\end{aligned}$$

We will estimate the gradient of this likelihood with samples from η. The resulting algorithm is exactly that of random dropouts. We propose that random dropouts help generalization performance because they provide better lower bounds to the noisy probabilistic models defined above, which in turn are better models of the sparse, 1/f distributions in real world data. It would be interesting to obtain better lower bounds to the likelihood of the noisy RNN models but our early attempts to sample the posterior $\mathrm{P}\left(\eta|\mathbf{y}_{t+1}, \mathbf{x}_t\right)$ were unsuccessful due to the high computational cost of the procedure.

We also found that column normalization (CN) helped to further increase the performance of the FPNN on the Penn corpus. CN consists of fixing the L2 norm of the incoming/outgoing weights to each hidden unit. In the FPNN we find that low values of the column norm result in longer timescales. This is because large values of the hidden units can only be obtained by having large self-connections. In turn, large values of the hidden units are needed to generate low entropy predictive distributions for each word. Notice CN was also used in [8] in conjunction with the random dropout method but apparently for different reasons. The authors of [8] found that CN helped them use larger gradient steps during learning, while we found that CN helped generalization. Unfortunately the magnitude to which the columns of $\mathbf{W}$ and $\mathbf{Z}$ are normalized does influence the performance of the model so we had to crossvalidate this parameter separately by running 5 different experiments to find a good value for the norm.

4 Results

We report the performance of the studied models on two datasets commonly used in the literature. The first is the Penn Corpus which contains 800k word tokens, the second is the 'text8' dataset which contains a cleaned up version of the first 100 million characters from Wikipedia [1]. Both of these corpora have well-defined cross-validation folds as described in [6], which allows us to directly compare our results to those reported elsewhere.

4.1 Word-level language modelling with linear RNNs

It is commonly believed that the recurrent neural network (RNN) based models are able to capture highly nonlinear distributions of sequential data which simpler feedforward NNs cannot, thus

[1] http://mattmahoney.net/dc/text8.zip

	Single train/test	+KN5+cache	x10	x10+KN5+cache
5-gram Kneser-Ney[1]	10/141.2	125.7		
feedforward NNLM[1]	?/140.2	106.6		
Log-bilinear LM[1]	?/144.5	105.8		
RNN[1]	?/124.7	97.5	102.1	89.4
dynamic RNN[1]	?/123.2	98.0	101.0	90.0
FPNN(no reg)	32/137			
FPNN(L1 reg)	?/125			
RNN[3] (no reg)	34/**126**			
RNN[3] ([2]DO&CN)	40/**107**			
FPNN ([2]DO&CN)	42/**102**	**94**	98.8	92.5
dynamic FPNN([2]DO&CN)	?/**98.4**	**90.7**	95.1	89.1

Table 1: Results on Penn corpus.
[1] as reported in [1], [2] trained with random dropouts and column normalization, [3] our implementation

explaining the better performance of RNNs for word-level language modelling. We find that this perspective has to be somewhat altered. In fact, the unregularized FPNN achieves similar perplexities on the training data with the unregularized nonlinear RNN, meaning it finds more patterns in the data. Instead, the RNN generalizes better and achieves lower perplexities on test data. This means the nonlinear nature of the RNN serves as a kind of regularizer, allowing the network to learn some important patterns present in language and preventing some spurious associations to be made, which the feedforward models make more easily. As a generative model, the RNN can be said to have better inductive biases than a feedforward neural network, perhaps owing to its dynamical representation and nonlinearities. However, after random dropout regularization, we find that the performance of the two models considered here increases significantly and levels off. The regularized FPNN still overfits more to the training data than the regularized RNN, but their predictive performances are nearly the same. The FPNN but not the RNN is further improved by column normalization. The regularized FPNN achieves the state-of-the-art result on the Penn corpus for a single model with a perplexity of 102.5 down from the previous best published result of 123. Mixtures of models usually fare better and can enhance performance to a perplexity of 77 when a large number of very different models are averaged together [1].

In the literature, it is common to also report the results of models averaged with KN5, unigram caches and/or averaged with multiple copies of the model trained independently. All of these further improve the results considerably as shown in table 2b. We highlight that of special interest are results for single models, or for single models combined with n-gram based models, as these will have the lowest computational cost at runtime and lowest memory trace. This cost can be significant with large vocabularies. Also, the Penn dataset is very small and on larger datasets training more than one model can be computationally expensive and memory taxing. One interesting aspect of the FPNN is that we can directly assess how large of a context the network uses by looking at the diagonal matrix $\mathbf{R}$. The effective contribution to $\mathbf{x}_t$ from a timelag $\tau > 0$ in the past is $\mathbf{R}^{\tau-1}\mathbf{W}\mathbf{y}^{t-\tau}$ which then decays like $\mathbf{R}^{\tau} = \exp(\log(\mathbf{R})\tau)$. The timescales of the network are defined as $-1/\log(R)$, where the division operation is understood elementwise on the diagonal of $\mathbf{R}$. For an FPNN trained on the Penn corpus we plot these timescales in figure 2. One can see that the FPNN has learned several very long timescales of up to 50 words. This reminds of the benefits offered by caching methods to language models: many language models are improved at test time if the predictive probability of the model is interpolated with a unigram model learned exclusively from the previous 50-100 words. We can see that nonetheless most of the timescales of the FPNN are relatively short in order to model the local grammar of language. Each experiment on the Penn dataset took about three hours to run on a GTX 680 GPU.

We also ran the same experiments on a much larger corpus known as 'text8', consisting of the first 100M characters of Wikipedia. As a preprocessing step we considered only words that appear at least five times in the training set, converting all others to the special '$<$ unk $>$' token. This left us with a large vocabulary of 67428 words and a training corpus of 15301600 words. Training RNN models on this much data would require several days for a single experiment so we turned to the fast approximate training method called noise contrastive estimation, proposed by [9] and adapted

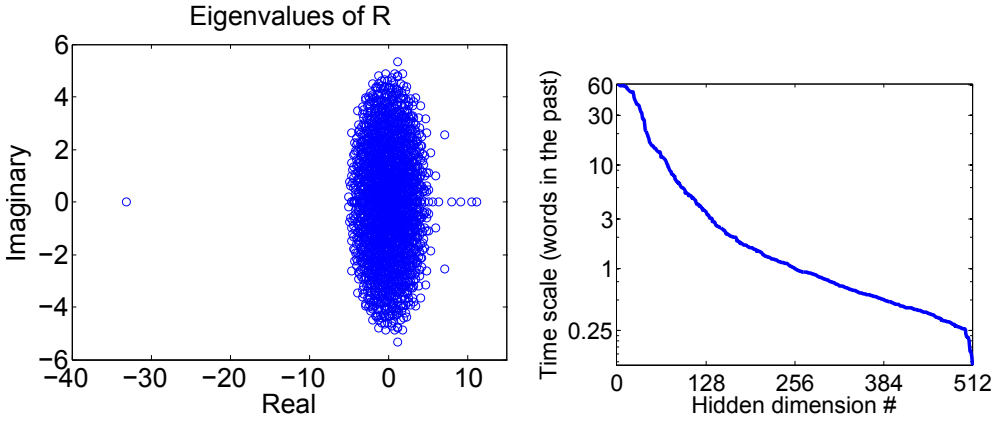

Figure 2: a. The eigenvalues of $\mathbf{R}$ for the RNN on character data. Notice that many eigenvalues are above 1 in absolute value, however the extra shrinkage from the pointwise rectifying nonlinearity will pull most of them inside the unit circle. The large negative eigenvalue corresponds to the stabilizing eigenvector. b. The eigenvalues of $\mathbf{R}$ for the FPNN on the Penn Corpus. The y-axis of this figure is logarithmically scaled. Notice that the FPNN has discovered long dependencies in language, as indicated by the long timescales of up to fifty words in $\mathbf{R}$. It is interesting that in our experiments with caching models, the unigram cache also only helps the likelihood up to a length of 50-100 words in the past. On the other hand, most timescales are rather very short, probably due to the strict syntactical regularities present in language.

for neural language modelling by [10]. The reader is advised to read [10] for full details of the procedure. Briefly, the method involves training a classifier to distinguish between real sampled sequences of words and sequences sampled from noise distributions, where the classifier is based on the generative model which we are training. We find that NCE trained RNNs produce as good results on Penn as those reported above with maximum likelihood estimation. On 'text8' we can no longer train the models by MLE, however the perplexity results we obtain are significantly better than n-gram models.

We find that dropout regularization no longer helps generalization with this large amount of training data. Instead, the performance of the FPNN and nonlinear RNN models is almost the same, and much better ($\approx 35\%$) than the vanilla n-gram model to which comparisons are usually made in the literature (Kneser-Ney of order 5). However, cache models significantly improve n-grams and in this case by a very large margin of $\approx 25\%$ perplexity. The neural language models offered improvements of 15% perplexity over that. Averaged together, the linear and nonlinear RNN further lowered perplexity by 20 points.

4.2 Character-level modelling with nonlinear RNNs

To redeem recurrent neural networks as useful language models, we point out that character-level language modelling is intrinsically more nonlinear and impossible to capture with linear and feed-forward neural networks. We show similar results to [6] and again report performance both on the training and testing datasets. It can be seen that the FPNN achieves very poor cross-entropy even on the training data. We experimented with adding a neural network nonlinearity in between the hidden representation $\mathbf{x}_t$ and the output which increased the likelihood but still left it very small. On the other hand, the SGD trained RNN successfully learns a language model on par with the Hessian-Free trained M-RNN reported in [6].

To our knowledge this is the first published result showing that stochastic gradient descent (SGD) learning of nonlinear RNNs can achieve the same performance as the second order optimization methods proposed by [7] and evaluated on the text8 dataset by [6]. The training cost of RNN with gradient descent is about 15h on a single high-end GPU card, while the HF-MRNN takes five days on 8 high-end GPUs, so we observe a more than 50 fold speedup for the same predictive likelihoods. Notice that while this very same architecture is evaluated in [5], their results are much worse, owing perhaps to the small size of the network they use (512 neurons as opposed to our 2048) and the

non-smooth version of the rectifier nonlinearity used there. We mention that for SGD learning to work well we had to use large momentum values (near 1). Clipping the gradients as done in [5] did not seem necessary. While we did observe sharp falls of the cost function a few times during training, these were not associated with large changes in the parameters, and the network recovered with a few parameter updates to its previous value of the cost function. Large momentum seemed however to be crucial for fast learning. Without it, the network diverged even with much smaller learning rates.

In separate experiments we trained the same RNN on the raw version of the same first 100 MB of Wikipedia. We found that SGD training diverged much more easily in that case and required clipping the gradients as is also done in [5]. On closer inspection, we found that all such cases of divergence were associated with very rare events not related to language at all, such as multiple repeats of the same low-probability character like dashes and exclamation marks. With a minimum of preprocessing we removed such events and the RNN was able to learn successfully from raw Wikipedia without clipping gradients in SGD. More research is needed to clarify the actual geometry of neural networks because as [5] points out, the classical image of long narrow ravines might be quite wrong.

To understand the model learned by the RNN we performed the eigendecomposition of the transformation matrix $\mathbf{R}$. While the Jacobian of the FPNN is just $\mathbf{R}$ for every sequential transformation, for the nonlinear network the Jacobian needs to be multiplied elementwise with the derivatives of the nonlinearity. Therefore by doing the eigendecomposition of $\mathbf{R}$ we do not directly obtain the timescales of the model. What we found was that the RNN discovered an ingenious solution to keeping itself stable while still using long timescales. As can be seen in the spectrum plotted in figure 2, $\mathbf{R}$ has many eigenvalues greater than 1 in absolute value, however one eigenvalue was in all our experiments particularly large and negative. We looked at the left and right eigenvectors associated with this eigenvalue and found they both mostly had all-positive or all-negative components for all the dimensions of $\mathbf{x}_t$. We call this the stabilizing eigenvector because it can be understood as a global inhibitor of the activity of the RNN. When many dimensions of $\mathbf{x}_t$ become large, their projections on the stabilizing right eigenvector add up which feeds back to the network through the left stabilizing eigenvector with an inverted sign from the negative eigenvalue. As a consequence, the net effect is for the network to be strongly and globally inhibited whenever too many dimensions of $\mathbf{x}_t$ activate strongly. Because our nonlinearity is rectifying, all neurons that are strongly inhibited end up on the negative branch of the nonlinearity which sets their activation to 0. We emphasize that this was an emergent property of the network that we did not anticipate and which is not immediately apparent under a superficial analysis of the matrix RNN. The same property emerged for the nonlinear RNN trained at the word-level.

It is much easier to extend RNNs when gradient descent is used for learning, as opposed to when the Hessian-Free method is used. This can be important; as shown in [6] character-level modelling generally lags behind word-level modelling in terms of predictive likelihoods so further work is needed to make character-level RNNs competitive. Here we present one enhancement we found with a simple extension of the RNN. We added skipping connections between the hidden units corresponding to the first letters of consecutive words. These connections formed a new matrix of parameters $\mathbf{R}_{\text{skip}}$ of the same size as $\mathbf{R}$. $\mathbf{R}_{\text{skip}}$ can potentially propagate information over longer distances in text. The skipping RNN is effectively a hybrid between a word-level and a character-level model. Adding in $\mathbf{R}_{\text{skip}}$ lowered the cross-entropy on the test set from 1.55 bpc to 1.48 bpc, while slightly increasing training times. This performance is still only about on par with state-of-the-art word-level n-gram models on this dataset, as estimated in [6].

However, character-level language modelling remains attractive for at least three reasons: being able to assign probability to any sequence of characters and even new words, the small number of parameters, and having a runtime that does not depend on the size of the vocabulary of words. As we have seen this third advantage might be nullified by the noise contrastive estimation method that we used above to obtain state-of-the-art results with word-level models. The second advantage might also be less major as we have seen that a very large number of parameters can be trained when good regularization methods are employed. The first advantage however seems important, as many highly inflected languages have vocabularies too large for word-level language models. Furthermore, there is important information in character-level descriptions of words, as words with the same stems or roots refer to similar abstract entities or events like 'rain', 'rains', 'rained', 'raining', 'rainy'. Also important is the fact that similar suffixes and prefixes indicate similar syntactical roles which again

	Bits per character
HF-MRNN[1]	1.54
RNN[2]	1.80
FPNN w/ NN output	2.12
RNN[3]	1.55
skipping H-RNN	1.48
subword ME-RNN[1]	1.46

(a) Character-level models.
[1] as reported in [6]. [2] as reported in [5]. [3] our implementation.

	Perplexity
KN5	298
KN5+cache	228
FPNN	197
RNN	191
FPNN + RNN	168

(b) Word-level models

Table 2: Results on text8.

can and should be used to generalize, for example from 'gloomy' to 'rainy' on the basis of the suffix '-y'. Although the performance of the RNN is modest on text8 compared to the word-level FPNN, the number of parameters differs by a factor of 20. It is thus still possible that a bigger nonlinear character-level RNN will perform better.

We note that a different solution for improving character-level language models has been discussed in [6], where the authors split words into their most frequent parts. These subparts consisted of short sequences of 1-3 characters. Indeed the HF-MRNN trained on the fragmented words achieved better cross-entropies that match the skipping RNN proposed here as can be seen in table 2a. To further improve on the model, [6] adds maximum entropy n-gram features to the network further lowering cross-entropy to 1.46 bpc. In general, [6] notices that neural language models can benefit from adding n-gram features directly to the predictive likelihoods. In line with this observation, we hypothesize that good subword language models can be achieved by separately predicting the most frequent words in a language and using a character-level model for the rest of the words. This remains a topic of future research.

5 Conclusions

We have shown here several new results on neural language modelling validated by empirical experiments. First, we have shown that the advantage of RNN based models over simpler linear and non-recurrent networks is due to their better implicit regularization rather than an ability to discover more complex nonlinear patterns. While the RNN had lower test perplexity on Penn corpus than the FPNN, it assigned higher perplexity to the training data, so it overfitted less than the FPNN. In fact, if perplexity on the training data is a measure of a model's flexibility, n-grams are by far the most flexible and nonlinear LMs. The objective of language modelling should be to design models that overfit less, while also having great flexibility and we say that such models have better inductive biases. This inductive bias can also be seen as a form of regularization. Neural LMs have some good inductive biases because different words with similar syntactic and semantic content can have similar latent representations and thus inherit each other's predictive properties. However, even among neural LMs we have seen that some models generalize better than others so it seems fruitful to look for neural models with better inductive bias.

As a second result presented in this work, we introduced a new class of neural LMs with better inductive biases by including noise in the generative process of the model. Alternatively, adding noise to the generation process can be seen as the random dropout regularization method recently introduced in [8]. Both the RNN and FPNN improve perplexity by $\approx$15% and $\approx$25% respectively on the test set of the Penn corpus which makes their performance about equal. The FPNN is further helped by a regularization technique of normalizing the input/output parameters to/from each hidden unit, making this highly regularized but linear LM the state-of-the-art single model on the popular Penn corpus. We also conducted experiments on a larger corpus known as 'text8', comprising of the first 100 MB of Wikipedia. This dataset is much larger than Penn, so we expected the benefits of regularization to be less important. In fact, with no regularization at all, RNNs and FPNNs performed comparably well and much better than vanilla n-grams. Dropout regularization did not help the LMs for the model sizes considered here (512 hidden units). It seems much larger LMs should be considered for further perplexity improvements on this large corpus. We emphasize that

this result questions the commonly held assumption that highly nonlinear functions are needed for state-of-the-art language models. We also showed that the NCE training method we use proposed in [9] and adapted to LM in [10] can in fact train state-of-the-art language models, which was not shown there.

To investigate a language problem where nonlinear RNNs might be necessary, we studied the character-level prediction task. As our third result we showed that the RNN achieves much better likelihoods on this problem than the FPNN, even on training data. The RNN's advantage on this problem is thus truly a consequence of its highly nonlinear nature. We also showed successful SGD training of RNNs at the character-level to performance levels achieved by the Hessian-Free method of [7], at less than a fiftieth of the computational cost. Crucial to successful SGD training was the use of very high momentum rates and the rectifier but smooth nonlinearity. SGD training is far easier to implement and extend than second-order methods, so we expect it should lead in the future to faster development of new neural LM architectures. As an example, we implemented a skipping RNN where additional connections are added between the first letters of consecutive words. This greatly increased the cross-entropy per character of the RNN to levels that matched word level n-grams. More work is needed to improve these promising character-level models to match the performance of the word-level neural LMs.

References

[1] T Mikolov, A Deoras, S Kombrink, L Burget, and JH Cernocky. Empirical evaluation and combination of advanced language modeling techniques. *Conference of the International Speech Communication Association*, 2011.

[2] A Mnih and G Hinton. Three new graphical models for statistical language modelling. *International Conference on Machine Learning*, 2007.

[3] S Hochreiter and J Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):17351780, 1997.

[4] S Hochreiter and J Schmidhuber. Learning long-term dependencies with gradient descent is diffcult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[5] R Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv:1211.5063v1*, 2012.

[6] T Mikolov, I Sutskever, A Deoras, HS Le, S Kombrink, and J Cernocky. Subword language modelling with neural networks. *unpublished*.

[7] I Sutskever, J Martens, and G Hinton. Generating text with recurrent neural networks,. *International Conference on Machine Learning*, 2011.

[8] GE Hinton, N Srivastava, A Krizhevsky, I Sutskever, and RR Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580v1*, 2012.

[9] M Guttmann and A Hyvarinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. *Proc Int Conf on Artificial Intelligence and Statistics (AISTATS2010)*, 2010.

[10] A Mnih and YW Teh. A fast and simple algorithm for training neural probabilistic language models. *International Conference on Machine Learning*, 2012.